Sheet 4 solution

1. a) The program displays a circle instead of a sphere. The reason for this is viewing setting. The program does not set any viewing setting. Hence, the viewing is set to the default in OpenGL which makes the viewing volume as a cube 2x2x2 centered at the origin. The contents of this cube are projected on the image plane using orthogonal projection. Since our sphere is centered at the origin, it appears as a circle instead of a sphere.
   b) By changing the angle steps, we control the approximation. Smaller angle steps should give smother approximation. The reverse is also true.

2. The following equations is used to convert from CMY subtractive color model to the RGB mode R=1.0-C, G=1.0-M, B=1-Y.

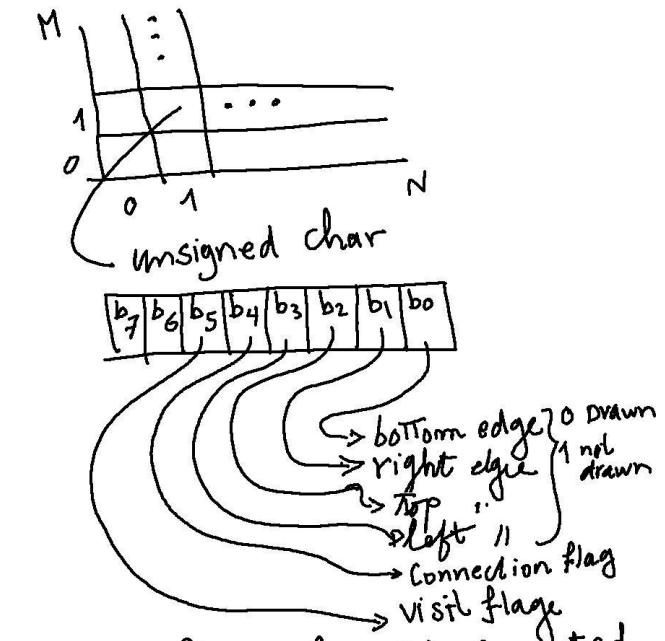
3.
```
#include "stdafx.h"
# include <glut.h>

void drawImage()
{
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);
glColor3f(1.0,0.0,0.0);
glVertex3f(-0.5,-0.5,0);
glColor3f(0.0,1.0,0.0);
glVertex3f(0.5,-0.5,0);
glColor3f(0.0,0.0,1.0);
glVertex3f(0,1.14-0.5,0);
glEnd();
glFlush();
}

int _tmain(int argc, _TCHAR* argv[])
{

    glutInitWindowSize(300,300);
    glutInitWindowPosition(300,300);
    glutCreateWindow("Maxwelle triangle");
    glutDisplayFunc(drawImage);
    glutMainLoop();
    return 0;
}
```

4.



Connection flag: 1 → the cell is connected to every other cell through open (not drawn) edges

visit flage: 1 → the cell is visited in the current connectivity check

Drawing the grid

we start with the cell (0,0) which must have a left and bottom edges closed because they are on the boundries. then we choos randomly if the right and bottom edge are open or not. if a cell right or bottom is on the boundry the made closed, otherwise it is set open or closed randomly

```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
// global variables
const int M=4,N=4;
GLfloat MazeCellSideLength=0.2F;
unsigned char Maze[M][N];
// thie function take a cell and a number the comes from
// the decimal of all bits in the cell are zeo except the bit corresponding
// to the edge to be randomized
unsigned char RadomizeCellEdge(unsigned char edgeBit,unsigned char cell)
{
    if(rand()%2>0)
        cell=cell | edgeBit;
    else
        cell=cell & !edgeBit;
    return cell;
}

void PrepareMaze(void)
{
    // initialize thz maze(rows increase up and columns increase right)
    // cell bits:
    // bit 0 bottom edge
    // bit 1 right edge
    // bit 2 top edge
    // bit 3 left edge
    // edge convension 0: closed, 1 open

    // First prepare a Maze in which each cell is open from at least one side
    for(int i=0;i<M;i++)
    {
        for(int j=0;j<N;j++)
        {
            // initially the cell is closed
            Maze[i][j]=0;
            if(j>0)// If there is a left neighbor
            {
                if((Maze[i][j-1]&0x02)>0)// if the cell left neighbor has its right edge open
                {
                    Maze[i][j]|=0X08 ;// open the cell left
                }
                else
                {
                    Maze[i][j]&=0XF7;//close the cell left
                }
            }
            if(i>0)// If there is a bottom neighbor
            {
                if((Maze[i-1][j]&0x04)>0)// if the cell bottom neighbor has its top edge open
                {
                    Maze[i][j]|=0X01 ;// open the cell bottom
                }
                else
                {
                    Maze[i][j]&=0XFE;//close the cell bottom
                }

            }
            while((Maze[i][j] & 0x0F) ==0)// repeat if closed cell
            {
                // if the last cells open its left or bottom edge
                if(i==M-1 && j==N-1)
                {
                    Maze[i][j]=Maze[i][j]|0X01;// bottom open
                    // make sur that thee neighbor is compatible
                    Maze[i-1][j]=Maze[i][j]|0X04;
                    break;
```

```
69                  }
70                  // set the right edge randomly
71                  if(j<(N-1)) Maze[i][j]=RadomizeCellEdge(0X02,Maze[i][j]);
72                  // set the top edge randomly
73                  if(i<(M-1)) Maze[i][j]=RadomizeCellEdge(0X04,Maze[i][j]);
74              }
75          }
76      }
77  }
78
79  void mydisplay()
80  {
81      // draw the Maze
82      glBegin(GL_LINES);
83      for(int i=0;i<M;i++)
84      {
85          for(int j=0;j<N;j++)
86          {
87              // if there is a left edge
88              if((Maze[i][j]&0X08) == 0)
89              {
90                  glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
91                  glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
92              }
93              // if there is a top edge
94              if((Maze[i][j]&0X04) == 0)
95              {
96                  glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
97                  glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
98              }
99              // if there is a right edge
100             if((Maze[i][j]&0X02) == 0)
101             {
102                 glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
103                 glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
104             }
105             // if there is a bottom edge
106             if((Maze[i][j]&0X01) == 0)
107             {
108                 glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
109                 glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
110             }
111         }
112     }
113     glEnd() ;
114     glFlush();
115 }
116 int main(int argc, char** argv){
117     PrepareMaze();
118     glutCreateWindow("Maze");
119     glutDisplayFunc(mydisplay);
120     glutMainLoop();
121 }
```

5.

```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
// global variables
const int M=4,N=4;
GLfloat MazeCellSideLength=0.2F;
unsigned char Maze[M][N];
// thie function take a cell and a number the comes from
// the decimal of all bits in the cell are zeo except the bit corresponding
// to the edge to be randomized
unsigned char RadomizeCellEdge(unsigned char edgeBit,unsigned char cell)
{
    if(rand()%2>0)
        cell=cell | edgeBit;
    else
        cell=cell & !edgeBit;
    return cell;
}
void PutConnectivityMarks(int row,int col)
{
    Maze[row][col]|=0X10;// make the current cell as connected
    Maze[row][col]|=0X20;// make the current cell as visited
    // first for not connected cell that can be reached now from the current cell
    // if there is a not connected  cell below that can be reached now by the current cell bottom edge
    if(row>0 && (Maze[row-1][col]&0X10)==0 && (Maze[row][col]&0X01)>0)
        PutConnectivityMarks(row-1,col);
    // if there is a not connected  cell to the left that can be reached now by the current cell left edge
    if(col>0 && (Maze[row][col-1]&0X10)==0 && (Maze[row][col]&0X08)>0)
        PutConnectivityMarks(row,col-1);
    // if there is a not connected  cell above that can be reached now by the current cell top eadge
    if(row<(M-1) && (Maze[row+1][col]&0X10)==0 && (Maze[row][col]&0X04)>0)
        PutConnectivityMarks(row+1,col);
    // if there is a not connected  cell to the right that can be reached now by the current cell right eadge
    if(col<(N-1)>0 && (Maze[row][col+1]&0X10)==0 && (Maze[row][col]&0X02)>0)
        PutConnectivityMarks(row,col+1);

    // second for connected cells that can be reached from the current cell but not visted yet
    // this condition can be included with the previous but will make it complex
    // if there is a connected  cell below that is not visited
    if(row>0 && (Maze[row-1][col]&0X10)>0 && (Maze[row-1][col]&0X20)==0)
        PutConnectivityMarks(row-1,col);
    // if there is a connected  cell to the left that is not visited
    if(col>0 && (Maze[row][col-1]&0X10)>0 && (Maze[row][col-1]&0X20)==0)
        PutConnectivityMarks(row,col-1);
    // if there is a connected  cell above that is not visited
    if(row<(M-1) && (Maze[row+1][col]&0X10)>0 && (Maze[row+1][col]&0X20)==0)
        PutConnectivityMarks(row+1,col);
    // if there is a connected  cell to the right that is not connected
    if(col<(N-1)>0 && (Maze[row][col+1]&0X10)>0 && (Maze[row][col+1]&0X20)==0)
        PutConnectivityMarks(row,col+1);
}
void MarkConnectivity(int row, int col)
{
    // reset visiting and connectivity marks
    for(int i=0;i<M;i++)
        for(int j=0;j<N;j++)
            Maze[i][j]&=0XCF;
    // put the connectivity marks starting from row , col
    PutConnectivityMarks(row,col);
}
void PrepareMaze(void)
{
    // initialize thz maze(rows increase up and columns increase right)
    // cell bits:
    // bit 0 bottom edge
```

```
66      // bit 1 right edge
67      // bit 2 top edge
68      // bit 3 left edge
69      // edge convension 0: closed, 1 open
70      // bit 4 connection flag (1 connected, 0 not connected)
71      // bit 5 visit flag (1 visited, 0 not visited)
72
73      // First prepare a Maze in which each cell is open from at least one side
74      for(int i=0;i<M;i++)
75      {
76          for(int j=0;j<N;j++)
77          {
78              // initially the cell is closed
79              Maze[i][j]=0;
80              if(j>0)// If there is a left neighbor
81              {
82                  if((Maze[i][j-1]&0x02)>0)// if the cell left neighbor has its right edge
    open
83                  {
84                      Maze[i][j]|=0X08 ;// open the cell left
85                  }
86                  else
87                  {
88                      Maze[i][j]&=0XF7;//close the cell left
89                  }
90              }
91              if(i>0)// If there is a bottom neighbor
92              {
93                  if((Maze[i-1][j]&0x04)>0)// if the cell bottom neighbor has its top edge
    open
94                  {
95                      Maze[i][j]|=0X01 ;// open the cell bottom
96                  }
97                  else
98                  {
99                      Maze[i][j]&=0XFE;//close the cell bottom
100                 }
101
102             }
103             while((Maze[i][j] & 0x0F) ==0)// repeat if closed cell
104             {
105                 // if the last cells open its left or bottom edge
106                 if(i==M-1 && j==N-1)
107                 {
108                     Maze[i][j]=Maze[i][j]|0X01;// bottom open
109                     // make sur that thee neighbor is compatible
110                     Maze[i-1][j]=Maze[i][j]|0X04;
111                     break;
112                 }
113                 // set the right edge randomly
114                 if(j<(N-1)) Maze[i][j]=RadomizeCellEdge(0X02,Maze[i][j]);
115                 // set the top edge randomly
116                 if(i<(M-1)) Maze[i][j]=RadomizeCellEdge(0X04,Maze[i][j]);
117             }
118         }
119     }
120     // second check connectivity
121     bool RemarkConnectivityNeeded=true;
122     while(RemarkConnectivityNeeded)
123     {
124         MarkConnectivity(0,0);
125         RemarkConnectivityNeeded=false;
126         for(int i=0;i<M;i++)
127         {
128             for(int j=0;j<N;j++)
129             {
130                 if((Maze[i][j]&0X10)==0)// if the cell is not connected
131                 {
132                     if(j>0) // if there is a cell at the left of the not connected
133                     {
```

```
134                         Maze[i][j-1]|=0X2;//open the right edge of the cell at left
135                         Maze[i][j]|=0X08;// open the left edge of the cell

136                     }
137                     else
138                     {
139                         //the cell can not be in the first row because we start with a connected cell
140                         // hence, there is always a connected cell below of the not connected cell at j=0
141                         Maze[i-1][j]|=0X04;//open the top edge of the cell below
142                         Maze[i][j]|=0X01;// open the bottom edge of the cell

143                     }
144                     RemarkConnectivityNeeded=true;
145                     break;
146                 }
147             }
148             if(RemarkConnectivityNeeded) break;
149         }
150     }
151     // open the first cell from left
152     Maze[0][0]|=0X08;
153     // open the last cell from right
154     Maze[M-1][N-1]|=0X02;
155 }
156
157 void mydisplay()
158 {
159     // draw the Maze
160     glBegin(GL_LINES);
161     for(int i=0;i<M;i++)
162     {
163         for(int j=0;j<N;j++)
164         {
165             // note that one edge may de drawn more than one (kept for clarity)
166             // if there is a left edge
167             if((Maze[i][j]&0X08) == 0)
168             {
169                 glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
170                 glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

171             }
172             // if there is a top edge
173             if((Maze[i][j]&0X04) == 0)
174             {
175                 glVertex3f(j*MazeCellSideLength,(i+1)*MazeCellSideLength,0);
176                 glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

177             }
178             // if there is a right edge
179             if((Maze[i][j]&0X02) == 0)
180             {
181                 glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);
182                 glVertex3f((j+1)*MazeCellSideLength,(i+1)*MazeCellSideLength,0);

183             }
184             // if there is a bottom edge
185             if((Maze[i][j]&0X01) == 0)
186             {
187                 glVertex3f(j*MazeCellSideLength,i*MazeCellSideLength,0);
188                 glVertex3f((j+1)*MazeCellSideLength,i*MazeCellSideLength,0);

189             }
190         }
191     }
192     glEnd() ;
193     glFlush();
194 }
195 int main(int argc, char** argv){

196     PrepareMaze();
197     glutCreateWindow("Maze");
198     glutDisplayFunc(mydisplay);
199     glutMainLoop();
200 }
```
